In the Specification:

The paragraph on page 1, spanning lines 4 through 12, has been amended as shown below:

1. R. Mech, et al., "Rendering Volumetric Fog and Other Gaseous Phenomena Using an Alpha Channel", U.S. Patent Application Serial No. 09/990,085 ~~(MS1-1032US)~~, filed concurrently herewith on November 21, 2001 and ~~(~~incorporated in its entirety herein by reference~~)~~; and

A1

2. R. Mech, "Method, System, and Computer Program Product for Rendering Multicolored Layered Fog with Self-Shadowing and Scene Shadowing", U.S. Patent Application Serial No. 09/990,082 ~~(MS1-1033US)~~, filed concurrently herewith on November 21, 2001 and ~~(~~incorporated in its entirety herein by reference~~)~~.

The paragraph spanning page 1, line 18 through page 2, line 4, has been amended as shown below:

A2

Computer graphics systems are used in many game and simulation applications. For example, flight simulators use graphics systems to present scenes that a pilot would be expected to see if he or she were flying in an actual aircraft cockpit. Several key techniques used to achieve visual realism include antialiasing, blending, polygon offset, lighting, texturizing, and atmospheric effects. Atmospheric effects such as fog, smoke, smog, and other gaseous phenomena are particularly useful because they create the effect of objects appearing and fading at a distance. This effect is what one would expect as a result of movement. Gaseous phenomena are especially important in flight simulation applications because they help ~~helps~~ to train pilots to operate in and respond to conditions of limited visibility.

The paragraph on page 3, spanning lines 4 through 16, has been amended as shown below:

In real-time animations, smoke and clouds are usually simulated by mapping transparent textures on a polygonal object that approximates the boundary of the gas. Although the texture may simulate different densities of the gas inside the 3D boundary and compute even the light scattering inside the gas, it does not change, when viewed from different directions, and it does not allow movement through the gas without sharp transitions. Consequently, these techniques are suitable for rendering very dense gases or gases ~~gasses~~ viewed from a distance. Other methods simplify their task by assuming constant density of the gas at a given elevation, thereby making it possible to use 3D textures to render the gas in real time. The assumption, however, prevents using the algorithm to render patchy fog. What is needed is a way to render in real-time, complex scenes that include patchy fog and other gaseous phenomena such that efficiency and high quality visual realism are achieved.

A3

The paragraph on page 4, spanning lines 16 and 17, has been amended as shown below:

FIG. 8 is a flowchart diagram describing an OPENGL® implementation of a routine for resetting pixel colors located outside of a fog region.

A4

The paragraph on page 4, spanning lines 18 through 20, has been amended as shown below:

A5

FIG. 9 is a flowchart diagram describing an OPENGL® implementation of a routine for determining travel distance through fog from a reference point to a pixel.

The paragraph on page 7, spanning lines 1 through 20, has been amended as shown below:

FIG. 1 illustrates a block diagram of an example computer architecture 100, which may be implemented in many different ways, in many environments, and on many different computers or computer systems. Architecture 100 includes six overlapping layers. Layer 110 represents a high level software application program. Example software application programs include visual simulation applications, computer games or any other application that could take advantage of computer generated graphics. Layer 120 represents a three-dimensional (3D) graphics software tool kit, such as OPENGL® PERFORMER, available from SGI, Mountain View, California. Layer 125 represents a graphics application program interface (API), which can include but is not limited to an OPENGL® software product, available from SGI. Layer 130 represents system support such as operating system and/or windowing system support. Examples of such support systems include UNIX®, Windows®, and LINUX® operating systems. Layer 135 represents firmware which can include proprietary computer code. Finally, layer 140 represents hardware, including graphics hardware. Hardware can be any hardware or graphics hardware including, but not limited to, a computer graphics processor (single chip or multiple chip), a specially designed computer, an interactive graphics machine, a gaming platform, a low end game system, a game console, a network architecture, server, et cetera. Some or all of the layers 110-140 of architecture 100 will be available in most commercially available computers.

A6

The paragraph spanning page 7, line 21 through page 8, line 6, has been amended as shown below:

In one exemplary ~~implemenation~~ implementation, a gaseous phenomena generator module 105 is provided. The gaseous phenomena generator module 105 provides control steps necessary to carry out routine 400 (described in detail below). The gaseous phenomena generator module 105 can be implemented in software, firmware, hardware, or in any combination thereof. As shown in FIG. 1, in one example implementation, gaseous phenomena generator module 105 is control logic (e.g., software) that is part of application layer 110 and provides control steps necessary to carry out routine 400. In alternative implementations, gaseous phenomena generator 105 can be implemented as control logic in any one of the layers 110-140 of architecture 100, or in any combination of layers 110-140 of architecture 100.

A7

The paragraph on page 8, spanning lines 21 through 24, has been amended as shown below:

Fog unit 225 can obtain either linear or ~~non-liner~~ non-linear fog color values. Blending unit 230 blends the fog color values and/or pixel values to produce a single pixel. The output of blending module 230 is stored in frame buffer 235. Display 240 can be used to display images or scenes stored in frame buffer 235.

A8

The paragraph on page 14, spanning lines 7 through 13, has been amended as shown below:

A9

In step 515, the pixel color values of those pixels located outside the fog objects are reset. To do so, the fog objects are rendered and the number of front and back faces behind each pixel are counted. In counting the front and back faces, one (1) is added to the stencil buffer for each back face and one (1) is subtracted for each front face that fails the depth test. Where the number of front and back faces are equal, the pixel color value is reset to zero. FIG. 8 illustrates an OpenGL® implementation of step 515.

The paragraph on page 14, spanning lines 14 through 21, has been amended as shown below:

A10

In step 805, the stencil buffer is initialized to zero (0). In step 810, writing into the color buffer is disabled. In step 815, the front faces are culled using the OpenGL® implementation cull mechanism. In step 820, the stencil operation is set to increment on depth test fail. In step 825, the fog objects are drawn. Because culling is set to front faces, only back faces will be drawn. In step 830, the stencil operation is set to decrement on depth test fail. In step 835, the back faces are culled using the OpenGL® implementation cull mechanism. In step 840, the fog objects are again drawn. Due to back face culling, only the front faces are drawn.

The paragraph on page 15, spanning lines 10 through 12, has been amended as shown below:

A11

In step 525, the distances from the reference point to the front faces are determined and subtracted from the color buffer. An OpenGL® implementation of this step will now be explained with reference to FIG. 9.

The paragraph on page 15, spanning lines 13 through 18, has been amended as shown below:

A12

In step 930, the OpenGL® implementation culling mechanism is used to cull back faces. In step 935, the blend function is set to reverse_subtract. Finally, in step 940, the fog objects are again drawn. Here, only the front faces will be drawn since back face culling is enabled. Updates to the depth buffer are disabled during steps 905-940.

The paragraph on page 16, spanning lines 1 through 3, has been amended as shown below:

A13

Routine 900 will now be described in detail. In step 910, fog color and pixel color are initialized. In step 915, the OpenGL® implementation culling mechanism is used to cull front faces.

The paragraph on page 16, spanning lines 9 through 11, has been amended as shown below:

A14

In step 525, the distances from the reference point to the front faces are determined and subtracted from the color buffer. An OpenGL® implementation of this step will now be explained with reference to FIG. 9.

The paragraph on page 16, spanning lines 12 through 15, has been amended as shown below:

A15

In step 930, the OpenGL® implementation culling mechanism is used to cull back faces. In step 935, the blend function is set to reverse_subtract. Finally, in step 940, the fog objects are again drawn. Here, only the front faces will be drawn since back face culling is enabled. Updates to the depth buffer are disabled during steps 905-940.

The paragraph on page 17, spanning lines 4 through 8, has been amended as shown below:

A16

The pixel map can store an arbitrary function, for example the exp2 function supported by the OpenGL® implementation (e.g., $1\text{-}f = 1\text{-}e^{-p \cdot p \cdot z \cdot z}$). Unfortunately, on most systems a full screen pixel copy is so slow that the frame rate can drop well below 10 fps. The solution is to use a smaller window size (e.g., 640 x 480) or a system with several graphics pipes.

The paragraph on page 17, spanning lines 9 through 12, has been amended as shown below:

A17

To avoid the use of pixel maps, linear dependency of the attenuation factor f on the distance, as in the case of linear fog in the OpenGL® implementation is necessary. If the travel distance z is multiplied by a value $s = (z_e - z_s)/(D \cdot r)$, where D is the distance for which the attenuation factor becomes 0, the value 1 - f is obtained.

The paragraph on page 18, spanning lines 3 through 8, has been amended as shown below:

A18

In step 1205, a blend factor is set. The blend factors used are ONE_MINUS_DST_COLOR, and CONSTANT_COLOR. In step 1210, a constant blend color is set equal to the color of the fog. If the OpenGL® implementation blend-color extension is not available the fog color is white. It is necessary to ensure that only pixels that are visible are drawn. Thus in step 1215, the depth test is set to EQUAL and in step 1220, stencil test is used to draw each pixel only once.

The paragraph on page 18, on line 13, has been amended as shown below:

A19

FIGS. 13A-F illustrate an OpenGL® implementation of steps 405-415.